APPENDIX 1

```
import java.io.*;
import java.net.*;
import java.awt.*;
import java.util.*;
import java.applet.*;
import netscape.javascript.*;

/**
 * Asynchronously pushing pages to browsers.
 * The data can be pushed explicitly or data can be sent
 * by reference using a URL.
 *
 * The PushData signifies that it is an applet that
 * is waiting for push events.
 *
 * <P>
 * <B> PARAMETERS <BR>
 *    destwin = Destination Window's Name. <BR>
 *    port = Port Number to connect to, default is the port of the originating Web server.
<BR>
 *    desc = One Word Description of PushData applet.<BR>
*/
public class PushData extends Applet implements Runnable
{
  protected int          port;
  protected Socket        sock;
  protected String        window = "_new";
  protected String        description;

  protected Thread  runner;
  protected boolean threadRunning;

  private DataInputStream in;
  private DataOutputStream out;

  private static int     defaultPort = -1;

  // parameters
  private String targetWindow = "destwin";
  private String portNumber   = "port";
  private String appletDesc   = "desc";

  // Javascript Object
  private JSObject mainwin;

  /*
   * Get all the necessary parameters.
   * <ul>
   * <li>port number
   * <li>destination window
   * <li>description
   * </ul>
  */
  public void init()
  {
    URL url;
    String str;

    try
    {
        str = getParameter( portNumber );
        if ( str != null )
            defaultPort= Integer.parseInt( str );
    }
    catch (NumberFormatException e){ };

    description = getParameter( appletDesc );
```

```
      window = this.getParameter( targetWindow );

      mainwin = JSObject.getWindow(this);
    }

    /**
     * Gets called when loaded.
    */
    public void start()
    {
        runner = new Thread( this );

        threadRunning = true;
        runner.start();
    }

    /**
     * Gets called when browser leaves window.
    */
    public void stop()
    {
        threadRunning = false;
        closeSocket();
    }

    /*
     * Send/Receive messages from the server.
    */
    public void run()
    {
      int avail;
      String data, message;

      while ( threadRunning )
      {
          // open the connection to the server
          openSocket();
          if ( !threadRunning )
              break;

          // notify anyone that the connection is open
          connectionOpen();

          try
          {
              // open datastreams
              in = new DataInputStream( sock.getInputStream() );
              out = new DataOutputStream( sock.getOutputStream() );

              // get the initial connection message
              message = getConnectionMessage();
              if ( message != null )
                  out.writeBytes( message );

              // send/receive messages
              for( ; threadRunning ; )
              {
                  if ( sock == null )   // connection was broken
                      break;

                  if ( in.available() == 0 )
                  {
                      Thread.sleep( 500 ); // check for data every 1/2 second
                      continue;
                  }

                  data = readAvailableData( in );
```

```
                if ( data == null )      // connection broken
                    break;

                processMessage( data.trim() );
            }
        }
        catch( Exception e )
        {
            System.out.println(e);
        }

        // notify that the connection was closed
        connectionClosed();
    }
}

/**
 * Gets called when browser is closed.
 */
public void destroy()
{
    closeSocket();
}

/**
 * Send a message to the server.
 *
 * @param String
 */
public void sendMessage( String msg )
{
    String data, message;

    try
    {
        out.writeBytes( msg );
    }
    catch ( Exception e )
    {
        System.out.println("Send Exception: " + e );
    }
}


/**
 * Process the message for this java applet.
 * This can be overridden by other push applets
 * to process the message differently.
 * This method will push a URL into the destination window.
 * If its just data, then we will send data to the window's
 * JavaScript function <b>putText</b>.
 *
 * @param String
 */
public void processMessage( String msg )
{
    int offset;
    URL url;
    String data;

    offset = msg.lastIndexOf( ' ' );
    if ( offset > 0 )
        data = msg.substring( 0, offset );
    else
        data = msg;

    try
    {
            url = new URL( data );
            getAppletContext().showDocument( url, window );
        }
        catch ( MalformedURLException mue )
        {
            Object[] args = { msg };
            mainwin.call( "putText", args );
        }
    }

    /*
     * This method will provide the connection
     * message to be sent to the server when we
     * are connected.  This method can be overriden
     * to provide your server specific protocol.
     *
     * @param String
    */
    public String getConnectionMessage()
    {
        return "PushData:connect:" + description;
    }

    /**
     * Called when connection is opened.
     * Override this method if you want to be notified
     * on connection open.
    */
    protected void connectionOpen()
    {
    }

    /**
     * Called when connection is closed.
     * Override this method if you want to be notified
     * on connection closed.
    */
    protected void connectionClosed()
    {
    }

    /*
     * open the socket.
    */
    protected void openSocket()
    {
      URL url;

      url = getCodeBase();

      while (threadRunning && url != null)   // loop until socket is created
      {
         try
         {
            if ( defaultPort > 0 )
                port = defaultPort;
            else
                port = url.getPort();
            sock = new Socket( url.getHost(), port );
            break;
         }
         catch(Exception e)
         {
             System.out.println( "Error During Socket Open" );
             System.out.println( e );
         }

         try
```

```
            {
                Thread.sleep(5000);
            }
            catch ( InterruptedException ie )
            {
            }
        }
    }

    /*
     * Close the socket.
    */
    protected void closeSocket()
    {
      try
      {
          if ( sock != null )
          {
              sock.close();
              sock = null;
          }
      }
      catch(Exception e)
      {
          System.out.println( "Error During Socket Close" );
          System.out.println( e );
      }
    }

    /**
     * Read all the data that currently can
     * be read off the pipe.  The data always
     * starts with a length and then the data.
     *
     * @param DataInputStream
     * @return String
    */
    private String readAvailableData( DataInputStream in )
                           throws IOException
    {
        int bytes, bytesRead=0;
        byte[] b;

        if ( in.available() > 0 )
        {
            bytes = in.readInt();

            b = new byte[ bytes ];
            while ( bytesRead < bytes )
            {
                bytesRead += in.read( b, bytesRead, bytes-bytesRead );
            }
        }
        else
            return null;

        return new String( b );
    }
}
```